PATENT ABSTRACTS OF JAPAN

(11)Publication number : 2002-351672
(43)Date of publication of application : 06.12.2002

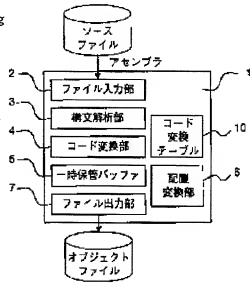
(51)Int.Cl. G06F 9/45

(21)Application number : 2001-156968
(22)Date of filing : 25.05.2001

(71)Applicant : MITSUBISHI ELECTRIC CORP
(72)Inventor : HIRATA AKIRA

(54) DEVICE AND METHOD FOR PROCESSING ASSEMBLER, AND PROGRAM FOR MAKING COMPUTER EXECUTE THE SAME METHOD

(57)Abstract:
PROBLEM TO BE SOLVED: To generate a program capable of performing the same processing even when it is operated by any of a big endian and a little endian.
SOLUTION: In this assembler processor 1 having an instruction set consisting of 16 bit length instruction and 32 bit length instruction and to generate an instruction code corresponding to a processor in which execution orders of two 16 bit length instruction codes are replaced when it is operated by a different endian, it is provided with a temporary storage buffer 5 to temporarily store the 16 bit length instruction code converted from a source file, an arrangement converting part 6 to replace the two 16 bit length instruction codes arranged in the temporary storage buffer 5 on the boundary of 32 bit and an file output part 7 to output the two replaced 16 bit length instruction codes to an object file.



(19)日本国特許庁（ＪＰ）　(12) **公開特許公報（Ａ）**　(11)特許出願公開番号
特開2002－351672
（P2002－351672A）
(43)公開日　平成14年12月６日(2002.12.6)

(51)Int.Cl.[7]	識別記号	ＦＩ	ﾃｰﾏｺｰﾄﾞ（参考）
Ｇ０６Ｆ　9/45		Ｇ０６Ｆ　9/44	３２０Ｄ　５Ｂ０８１

審査請求　未請求　請求項の数７　ＯＬ　（全 14 頁）

(21)出願番号　特願2001－156968(P2001－156968)

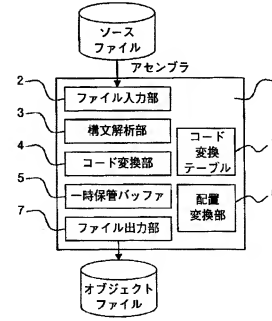
(22)出願日　平成13年５月25日(2001.5.25)

(71)出願人　000006013
三菱電機株式会社
東京都千代田区丸の内二丁目２番３号
(72)発明者　平田　明
東京都千代田区丸の内二丁目２番３号　三菱電機株式会社内
(74)代理人　100089118
弁理士　酒井　宏明

Ｆターム(参考)　5B081 AA07 DD01

(54)【発明の名称】　アセンブラ処理装置、アセンブラ処理方法およびその方法をコンピュータに実行させるプログラム

(57)【要約】
【課題】　ビッグエンディアン、リトルエンディアンのどちらのエンディアンで動作しても同一の処理を行えるプログラムを生成すること。
【解決手段】　１６ビット長命令と３２ビット長命令からなる命令セットを有するとともに異なるエンディアンで動作する時に２個の１６ビット長命令コードの実行順序が入れ替わるプロセッサに対応した命令コードを生成するアセンブラ処理装置１において、ソースファイルからコード変換された１６ビット長命令コードを一時的に格納する一時保管バッファ５と、一時保管バッファ５に３２ビット境界で配置された２個の１６ビット長命令コードを、動作対象のエンディアンに応じて入れ替える配置変換部６と、入れ替えられた２個の１６ビット長命令コードを、オブジェクトファイルへ出力するファイル出力部７とを備えた。



【特許請求の範囲】

【請求項１】　１６ビット長命令と３２ビット長命令からなる命令セットを有するとともに異なるエンディアンで動作する時に２個の１６ビット長命令コードの実行順序が入れ替わるプロセッサに対応した命令コードを生成するアセンブラ処理装置において、

ソースファイルからコード変換された１６ビット長命令コードを一時的に格納する記憶手段と、

前記記憶手段に３２ビット境界で配置された２個の１６ビット長命令コードを、動作対象のエンディアンに応じて入れ替える配置変換手段と、

入れ替えられた２個の１６ビット長命令コードを、オブジェクトファイルへ出力するファイル出力手段と、を備えたことを特徴とするアセンブラ処理装置。

【請求項２】　前記配置変換手段は、１６ビット長命令コードが前記記憶手段に格納されたときに、更に１６ビット長のＮＯＰ命令コードを前記記憶手段に追加するものであることを特徴とする請求項１に記載のアセンブラ処理装置。

【請求項３】　１６ビット長命令と３２ビット長命令からなる命令セットを有するとともに異なるエンディアンで動作する時に２個の１６ビット長命令コードの実行順序が入れ替わるプロセッサに対応した命令コードを生成するアセンブラ処理装置において、

２個の１６ビット長命令を組み合わせて１個の３２ビット長命令として定義した複合命令のニーモニックと、前記複合命令の命令コードとを対応づけたコード変換テーブルと、

ソースファイルから読み出した前記複合命令を、前記コード変換テーブルを参照して前記複合命令に対応する命令コードに変換するコード変換手段と、

を備えたことを特徴とするアセンブラ処理装置。

【請求項４】　１６ビット長命令と３２ビット長命令からなる命令セットを有するとともに異なるエンディアンで動作する時に２個の１６ビット長命令コードの実行順序が入れ替わるプロセッサに対応した命令コードを生成するアセンブラ処理方法において、

ソースファイルからコード変換された１６ビット長命令コードを、記憶手段に一時的に格納する記憶ステップと、

前記記憶手段に３２ビット境界で配置された２個の１６ビット長命令コードを、動作対象のエンディアンに応じて入れ替える配置変換ステップと、

入れ替えられた２個の１６ビット長命令コードを、オブジェクトファイルへ出力するファイル出力ステップと、を含むことを特徴とするアセンブラ処理方法。

【請求項５】　前記配置変換ステップは、１６ビット長命令コードが前記記憶手段に格納されたときに、更に１６ビット長のＮＯＰ命令コードを前記記憶手段に追加するものであることを特徴とする請求項４に記載のアセンブラ処理方法。

【請求項６】　１６ビット長命令と３２ビット長命令からなる命令セットを有するとともに異なるエンディアンで動作する時に２個の１６ビット長命令コードの実行順序が入れ替わるプロセッサに対応した命令コードを生成するアセンブラ処理方法において、

２個の１６ビット長命令を組み合わせて１個の３２ビット長命令として定義した複合命令のニーモニックと前記複合命令の命令コードとを対応づけたコード変換テーブルを参照して、ソースファイルから読み出した前記複合命令を前記複合命令に対応する命令コードに変換するコード変換ステップを含むことを特徴とするアセンブラ処理方法。

【請求項７】　請求項４～６のいずれか一つに記載されたアセンブラ処理方法をコンピュータに実行させるプログラム。

【発明の詳細な説明】

【０００１】

【発明の属する技術分野】この発明はソースファイルからプロセッサが実行する命令コードからなるオブジェクトファイルを生成するアセンブラ処理装置、アセンブラ処理方法およびその方法をコンピュータに実行させるプログラムに関するものであり、特にバイエンディアン・プロセッサで実行される命令コード（機械語）を生成するアセンブラ処理装置、アセンブラ処理方法およびその方法をコンピュータに実行させるプログラムに関する。

【０００２】

【従来の技術】図１８は、ソースファイルを入力しオブジェクトファイルに変換して出力する従来のアセンブラの構成を示すブロック図である。従来のアセンブラは、図１８に示すように、ファイル入力部１２と、構文解析部１３と、コード変換部１４と、オブジェクトファイル出力部１７とを備えた構成となっている。また、図１９は、従来のアセンブラによるアセンブラ処理のフローチャートであり、図１４はコード変換部１４で参照されるコード変換テーブル１０の一部を示す説明図である。ここで、図１４において、コード変換テーブル１０は、アセンブリ言語のニーモニック１８と各ニーモニックから変換される機械語（命令コード）１９とを対応させたテーブルである。

【０００３】図１９に示すように、従来のアセンブラでは、ソースファイルをファイル入力部１２から読み込み、読み込んだソースファイルに記述された命令を構文解析部１３によって解析する。そして、解析を行った後、コード変換部１４がコード変換テーブル１０を参照することによって、ニーモニックから機械語（命令コード）に変換する。変換された機械語は、ファイル出力部１７によってファイルへ順次出力してオブジェクトファイルを生成するようになっている。

【０００４】ビッグエンディアンとリトルエンディアン

とを切り替えて動作可能なプロセッサが近年登場している。ここで、ビッグエンディアンとは、データの最上位バイトがメモリ上の最下位アドレスに割り当てられるバイト順序付け方式であり、リトルエンディアンとはデータの最下位バイトがメモリ上の最下位アドレスに割り当てられるバイト順序付け方式のことをいう。プロセッサが双方のエンディアン（バイエンディアン）に対応している場合、ビッグエンディアン、リトルエンディアンのどちらで動作する場合でも、プロセッサで実行する機械語そのものはエンディアンに関係なく共通するものである。バイエンディアン対応の従来のアセンブラは、機械語をオブジェクトファイルに出力する際に機械語や扱うデータはそれぞれのエンディアンに従ったバイトの並びに変えてメモリイメージとして出力する。

【０００５】図１５はニーモニックを機械語に変換した時の１６進数表記のコードとメモリに配置したときの状態を示す説明図である。図１５では、アセンブリ言語で記述されたニーモニック２０と、ニーモニック２０に対応する機械語（命令コード）２１と、各命令がメモリ上に配置された時のアドレス２２との関係を示している。また、図１５では、命令Ａは３２ビット長の命令、命令Ｂ，Ｃは１６ビット長の命令を表し、命令Ａ，Ｂ，Ｃの順に実行するように記述したプログラムを一例として示している。

【０００６】図１６と図１７は、このプログラムをアセンブラ１１を用いて機械語に変換したコードをオブジェクトファイルとして出力し、このオブジェクトファイルの機械語をプログラムを実行するシステムのメモリ上に展開した時のメモリイメージを示す説明図である。図１６はビッグエンディアン動作を行う場合のメモリ上での命令コードの配置２３を示し、図１７はリトルエンディアン動作を行う場合でのメモリ上での命令コードの配置２４を表している。どちらの状態においてもアセンブラで生成した命令コードは、アドレス０番地に命令Ａ、アドレス４番地に命令Ｂ、アドレス６番地に命令Ｃが配置され、通常のプロセッサはアドレス順に、命令Ａ、命令Ｂ、命令Ｃの順序で実行を行う。

【０００７】

【発明が解決しようとする課題】ビッグエンディアンで動作し１６ビットと３２ビットの２種類の命令長を持つプロセッサが従来から一般的に知られている。図１２はこの種のプロセッサの命令コードを示した模式図である。また、図１３は、このプロセッサの動作を示すフローチャートである。図１３に示す通り、プロセッサは、オブジェクトファイルから３２ビット単位でデータを取り込む。そして取り込んだデータが３２ビット長の命令コードである場合にはその命令コードを実行する。一方、取り込んだデータが２命令からなる１６ビット長の命令コードであった場合には上位１６ビットの命令コード（図１２中、１６ｂｉｔ命令１）を先に実行し、下位１６ビット（図１２中、１６ｂｉｔ命令２）の命令コードを後に実行する。また、下位１６ビットの最上位ビットを同時実行フラグビットとし、このビットが１であった場合は読み込んだ２つの命令コードを並列に実行し、０であった場合には上述のように２つの１６ビット長命令コードを順次実行するようになっている。

【０００８】このようなビッグエンディアンで動作するプロセッサのバス・インタフェースに修正を加え、従来と異なるリトルエンディアンで動作できるようにしたプロセッサがある。このような修正を加えたプロセッサにおいても図１３のフローチャートと同様に命令実行の処理を行っている。

【０００９】このプロセッサがビッグエンディアンで動作した場合、図１６に示したメモリイメージの命令を実行すると、０番地の３２ビット［０ｘ９０Ｆ０１２３４］を読出し命令Ａとして実行する。つぎに、４番地の３２ビット［０ｘ００Ｃ２７Ｆ２０］を読出し、４番地の命令Ｂ、６番地の命令Ｃの順で実行する。

【００１０】一方、プロセッサがリトルエンディアンで動作した場合は図１７に示したメモリイメージにおいて、０番地の３２ビット［０ｘ９０Ｆ０１２３４］を読出し、命令Ａとして実行する。つぎに、４番地の３２ビット［０ｘ７Ｆ２０００Ｃ２］を読出し、６番地の命令Ｃ、４番地の命令Ｂの順で実行する。

【００１１】このように、プロセッサでは１６ビット長の２つの命令コードを順次実行した場合、ビッグエンディアンとリトルエンディアンとで命令の実行順が逆となり、リトルエンディアンの動作ではアドレス上で逆行する動作となる。

【００１２】このため、リトルエンディアン動作において以上の動作を行うプロセッサで実行するプログラムを作成する場合には、２つの１６ビット長命令コードを順次実行したときにアドレスと逆行することを意識したコーディングを行う必要がある。しかしながら、アセンブリ言語で記述する場合、使用する命令が３２ビット長であるか１６ビット長であるかを意識して記述することはないため、このような制限下でプログラムを記述することは困難であり、ソフトウェアの生産効率が悪くなる問題点があった。

【００１３】この発明は上記に鑑みてなされたもので、どちらのエンディアンで動作しても同一の処理を行うプログラムを生成でき、アセンブリ言語プログラム開発を効率よく行って、ソフトウェアの生産効率を向上させることができるアセンブラ処理装置を得ることを目的とする。

【００１４】

【課題を解決するための手段】上記目的を達成するため、この発明にかかるアセンブラ処理装置は、１６ビット長命令と３２ビット長命令からなる命令セットを有するとともに異なるエンディアンで動作する時に２個の１

６ビット長命令コードの実行順序が入れ替わるプロセッサに対応した命令コードを生成するアセンブラ処理装置において、ソースファイルからコード変換された１６ビット長命令コードを一時的に格納する記憶手段と、前記記憶手段に３２ビット境界で配置された２個の１６ビット長命令コードを、動作対象のエンディアンに応じて入れ替える配置変換手段と、入れ替えられた２個の１６ビット長命令コードを、オブジェクトファイルへ出力するファイル出力手段とを備えたことを特徴とする。

【００１５】この発明によれば、ソースファイルからコード変換された１６ビット長命令コードを記憶手段に一時的に格納し、記憶手段に３２ビット境界で配置された２個の１６ビット長命令コードを動作対象のエンディアンに応じて入れ替えて、２個の１６ビット長命令コードを、オブジェクトファイルへ出力するので、プロセッサがビッグエンディアンとリトルエンディアンのどちらのエンディアンで動作する場合でも、アセンブリ言語記述の同一ソースファイルを用いて同一の処理を実行するオブジェクトファイルを生成することができ、ソフトウェアの生産性を改善することができる。

【００１６】つぎの発明にかかるアセンブラ処理装置は、上記の発明において、前記配置変換手段は、１６ビット長命令コードが前記記憶手段に格納されたときに、更に１６ビット長のＮＯＰ命令コードを前記記憶手段に追加するものであることを特徴とする。

【００１７】この発明によれば、１６ビット長命令コードが記憶手段に格納されたときに、更に１６ビット長のＮＯＰ命令コードを記憶手段に追加するので、全ての１６ビット長命令を３２ビットの境界に配置して、リトルエンディアン動作時において１６ビット長命令がアドレスに逆行した実行動作となることを無視することができる。また、逆アセンブルでメモリイメージとニーモニックの対応表示を行った場合にはコーディングした手順がそのまま出力されるため、検証には都合がよくソフトウェア開発の効率化を行うことができる。

【００１８】つぎの発明にかかるアセンブラ処理装置は、１６ビット長命令と３２ビット長命令からなる命令セットを有するとともに異なるエンディアンで動作する時に２個の１６ビット長命令コードの実行順序が入れ替わるプロセッサに対応した命令コードを生成するアセンブラ処理装置において、２個の１６ビット長命令を組み合わせて１個の３２ビット長命令として定義した複合命令のニーモニックと、前記複合命令の命令コードとを対応づけたコード変換テーブルと、ソースファイルから読み出した前記複合命令を、前記コード変換テーブルを参照して前記複合命令に対応する命令コードに変換するコード変換手段と、を備えたことを特徴とする。

【００１９】この発明によれば、２個の１６ビット長命令を組み合わせた１個の３２ビット長命令を複合命令として定義するとともに、この複合命令のニーモニックと命令コードとを対応づけたコード変換テーブルを定義しておく。そして、このコード変換テーブルを用いて、ソースファイルから読み出した複合命令を複合命令の命令コードにコード変換するので、生成される命令コードはすべて３２ビット長となる。このため、プロセッサのリトルエンディアン動作時における命令の逆転実行を確実になくすことができ、また、コード量の最適化を図ることができる。

【００２０】つぎの発明にかかるアセンブラ処理方法は、１６ビット長命令と３２ビット長命令からなる命令セットを有するとともに異なるエンディアンで動作する時に２個の１６ビット長命令コードの実行順序が入れ替わるプロセッサに対応した命令コードを生成するアセンブラ処理方法において、ソースファイルからコード変換された１６ビット長命令コードを、記憶手段に一時的に格納する記憶ステップと、前記記憶手段に３２ビット境界で配置された２個の１６ビット長命令コードを、動作対象のエンディアンに応じて入れ替える配置変換ステップと、入れ替えられた２個の１６ビット長命令コードを、オブジェクトファイルへ出力するファイル出力ステップとを含むことを特徴とする。

【００２１】この発明によれば、ソースファイルからコード変換された１６ビット長命令コードを記憶手段に一時的に格納し、記憶手段に３２ビット境界で配置された２個の１６ビット長命令コードを動作対象のエンディアンに応じて入れ替えて、２個の１６ビット長命令コードを、オブジェクトファイルへ出力するので、プロセッサがビッグエンディアンとリトルエンディアンのどちらのエンディアンで動作する場合でも、アセンブリ言語記述の同一ソースファイルを用いて同一の処理を実行するオブジェクトファイルを生成することができ、ソフトウェアの生産性を改善することができる。

【００２２】つぎの発明にかかるアセンブラ処理方法は、上記の発明において、前記配置変換ステップは、１６ビット長命令コードが前記記憶手段に格納されたときに、更に１６ビット長のＮＯＰ命令コードを前記記憶手段に追加するものであることを特徴とする。

【００２３】この発明によれば、１６ビット長命令コードが記憶手段に格納されたときに、更に１６ビット長のＮＯＰ命令コードを記憶手段に追加するので、全ての１６ビット長命令を３２ビットの境界に配置して、リトルエンディアン動作時において１６ビット長命令がアドレスに逆行した実行動作となることを無視することができる。また、逆アセンブルでメモリイメージとニーモニックの対応表示を行った場合にはコーディングした手順がそのまま出力されるため、検証には都合がよくソフトウェア開発の効率化を行うことができる。

【００２４】つぎの発明にかかるアセンブラ処理方法は、１６ビット長命令と３２ビット長命令からなる命令セットを有するとともに異なるエンディアンで動作する

時に２個の１６ビット長命令コードの実行順序が入れ替わるプロセッサに対応した命令コードを生成するアセンブラ処理方法において、２個の１６ビット長命令を組み合わせて１個の３２ビット長命令として定義した複合命令のニーモニックと前記複合命令の命令コードとを対応づけたコード変換テーブルを参照して、ソースファイルから読み出した前記複合命令を前記複合命令に対応する命令コードに変換するコード変換ステップを含むことを特徴とする。

【００２５】この発明によれば、２個の１６ビット長命令を組み合わせた１個の３２ビット長命令を複合命令として定義するとともに、この複合命令のニーモニックと命令コードとを対応づけたコード変換テーブルを定義しておく。そして、このコード変換テーブルを用いて、ソースファイルから読み出した複合命令を複合命令の命令コードにコード変換するので、生成される命令コードはすべて３２ビット長となる。このため、プロセッサのリトルエンディアン動作時における命令の逆転実行を確実になくすことができ、また、コード量の最適化を図ることができる。

【００２６】つぎの発明にかかるプログラムは、上述した発明のいずれか一つに記載されたアセンブラ処理方法をコンピュータに実行させるプログラムであり、そのプログラムが機械読み取り可能となり、これによって、上記の発明のいずれか一つの動作をコンピュータによって実行することができる。

【００２７】

【発明の実施の形態】以下に添付図面を参照して、この発明にかかるアセンブラ処理装置、アセンブラ処理方法およびその方法をコンピュータに実行させるプログラムの好適な実施の形態を詳細に説明する。

【００２８】実施の形態１．図１は、この発明の実施の形態１であるアセンブラの構成を示すブロック図である。実施の形態１のアセンブラは、アセンブラ言語で記述されたソースファイルからビッグエンディアン用の命令コード（機械語）とリトルエンディアン用の命令コード（機械語）をオブジェクトファイルとして出力するバイエンディアン対応アセンブラである。また、実施の形態１のアセンブラは、上述した図１２に示す命令コードを有し、かつ上述した図１３のフローチャートに示す動作を行うプロセッサで実行される命令コードを生成するものである。

【００２９】図１に示すように、実施の形態１のバイエンディアン対応アセンブラ１は、ソースファイルを読み出すファイル入力部２と、ファイル入力部２で読み出したソースファイルに記述されたアセンブリ言語のソースコードを構文解析する構文解析部３と、コード変換テーブル１０と、コード変換テーブル１０を参照して、アセンブリ言語のニーモニックを命令コード（機械語）に変換するコード変換部４と、変換された命令コードを一時的に記憶する一時保管バッファ５と、一時保管バッファ５に保存された命令コードの配置を変換する配置変換部６と、変換された命令コードまたは一時保管バッファ５に記憶されている変換された命令コードを、オブジェクトファイルとして出力処理を行うファイル出力部７とを備えている。

【００３０】コード変換テーブル１０は、アセンブラ言語のニーモニック表記と命令コードとを対応付けた表であり、図１４はコード変換テーブル１０の一例を示す説明図である。一時保管バッファは、アセンブラ１が実行されるときに、記憶手段としてのメモリ上に４個のバイトデータの配列（ｃｈａｒ　ＢＵＦ［３］）を確保される。配置変換部６は、一時保管バッファ５に記憶された命令コードの配置を、動作するエンディアンに応じてバイト単位で変換するものである。

【００３１】つぎに、以上のように構成されたバイエンディアン対応アセンブラによる、ビッグエンディアン用の命令コード生成の処理手順について説明する。図２は、ビッグエンディアン用命令コード生成の処理手順を示すフローチャートである。

【００３２】アセンブラ１は、まず、一時保管バッファ５としてメモリ上に４個のバイトデータの配列（ｃｈａｒ　ＢＵＦ［３］）を確保して、これを初期化する（ステップＳ２０１）。そして、ファイル入力部２によりアセンブリ言語で記述されたソースファイルを一行分読み込み（ステップＳ２０２）、読み込んだソースコードを構文解析部３によって構文解析する（ステップＳ２０３）。構文解析終了後、コード変換部４においてニーモニックと命令コード（機械語）の対応表が記されているコード変換テーブル１０（図１４）を参照してソースコードを命令コード（機械語）に変換する（ステップＳ２０４）。

【００３３】そして、配置変換部６では、更に以下のビッグエンディアン用の命令コード生成のための配置変換処理を行う。まず、コード変換された命令コードのサイズをチェックする（ステップＳ２０５）。命令コードが３２ビット長であった場合には、更に一時保管バッファ５に前回変換分の命令コードが残存しているか否かをチェックする（ステップＳ２０６）。残存していなければ、変換された命令コードの上位バイトから１バイトずつ４バイト分を、一時保管バッファ５にＢＵＦ［０］、ＢＵＦ［１］、ＢＵＦ［２］、ＢＵＦ［３］の順で格納する（ステップＳ２０９）。そして、ファイル出力部７によって、一時保管バッファ５に格納された命令コードをオブジェクトファイルに出力する（ステップＳ２１３）。出力する順番及びバイトごとの配置は、ＢＵＦ［０］、ＢＵＦ［１］、ＢＵＦ［２］、ＢＵＦ［３］の順番で、ビッグエンディアンに対応させてオブジェクトファイルの下位アドレス（即ち、ＭＳＢ）から出力する。

【００３４】ステップＳ２０５におけるコードサイズのチェックの結果、変換された命令コードが１６ビット長であった場合は、一時保管バッファ５に前回変換分の命令コードが残存しているか否かをチェックする（ステップＳ２１０）。残存していなければ、１６ビット長命令コードの上位８ビットを一時保管バッファ５のＢＵＦ［０］に、下位８ビットをＢＵＦ［１］に保管する（ステップＳ２１１）。

【００３５】つぎに、ファイル入力部２においてソースファイルからつぎの一行のソースコードを読み出し（ステップＳ２０２）、同様にステップＳ２０５までの処理を行う。この命令が１６ビット長命令コードであった場合、一時保管バッファ５に前回変換分の命令コードが残存しているか否かをチェックするが（ステップＳ２１０）、この場合、ＢＵＦ［０］とＢＵＦ［１］に残存しているので、新たに変換した命令コードの上位８ビットをＢＵＦ［２］に、下位８ビットをＢＵＦ［３］に保存する（ステップＳ２１２）。そして、ファイル出力部７によって、一時保管バッファ５に格納された命令コードを、ＢＵＦ［０］、ＢＵＦ［１］、ＢＵＦ［２］、ＢＵＦ［３］の順番で、ビッグエンディアンに対応させてオブジェクトファイルの下位アドレス（即ち、ＭＳＢ）から出力する（ステップＳ２１３）。

【００３６】ステップＳ２０５で、１６ビット長命令コードのつぎに読み出して変換された命令コードが３２ビット長命令であった場合には、一時保管バッファ５に前回変換分の命令コードが残存しているか否かをチェックするが（ステップＳ２０６）、この場合、ＢＵＦ［０］とＢＵＦ［１］に残存しているので、ステップＳ２０７へ進み、一時保管バッファ５のＢＵＦ［２］、ＢＵＦ［３］に１６ビットの無処理命令（ＮＯＰ）を追加保存する。そして、ファイル出力部７によって、一時保管バッファ５に格納された命令コードを、ＢＵＦ［０］、ＢＵＦ［１］、ＢＵＦ［２］、ＢＵＦ［３］の順番で、ビッグエンディアンに対応させてオブジェクトファイルの下位アドレス（即ち、ＭＳＢ）から出力し（ステップＳ２０８）、続いて次の３２ビット長命令コードの一時保管バッファ５への保存（ステップＳ２０９）およびオブジェクトファイルへの出力を行う（ステップＳ２１３）。

【００３７】図４（ａ）は、一例としてソースコードが３２ビット長の命令コードＡ，１６ビット長の命令コードＢ，および１６ビット長の命令コードＣにこの順番でコード変換された場合の、ビッグエンディアン用の各命令コードの一時保管バッファ５への格納状態、およびオブジェクトファイルへの出力状態を示す説明図である。図４（ｂ）は、一例としてソースコードが１６ビット長の命令コードＢ，３２ビット長の命令コードＡにこの順番でコード変換された場合の、各命令コードの一時保管バッファ５への格納状態、およびオブジェクトファイルへの出力状態を示す説明図である。

【００３８】プロセッサがビッグエンディアンで動作する場合、プロセッサがオブジェクトファイルからメモリ上にロードされた命令コードを０番地、４番地、６番地の順に実行していくが、図４（ａ）および（ｂ）からわかるように、オブジェクトファイルへの各命令コードの格納配置はビッグエンディアンの場合に命令コードの実行順に配置されるので変換されるので、ソースファイルの命令の順番と命令の実行順が一致したものとなる。

【００３９】つぎに、実施の形態１のアセンブラ１によるリトルエンディアン用の命令コード生成の処理手順について説明する。図３は、リトルエンディアン用命令コード生成の処理手順を示すフローチャートである。アセンブラ１は、ビッグエンディアン用命令コード生成処理との場合と同様に、一時保管バッファ５の初期化（ステップＳ３０１）、ソースファイルから一行読出し（ステップＳ３０２）、構文解析（ステップＳ３０３）、コード変換を行う（ステップＳ３０４）。そして配置変換部６では、更に以下のリトルエンディアン用の命令コード生成のための配置変換処理を行う。まず、コード変換された命令コードのサイズをチェックする（ステップＳ３０５）。命令コードが３２ビット長であった場合には、更に一時保管バッファ５に前回変換分の命令コードが残存しているか否かをチェックする（ステップＳ３０６）。残存していなければ、コード変換された命令コードの下位バイトから１バイトずつ４バイト分を、一時保管バッファ５にＢＵＦ［０］、ＢＵＦ［１］、ＢＵＦ［２］、ＢＵＦ［３］の順で格納する（ステップＳ３０９）。そして、ファイル出力部７によって、一時保管バッファ５に格納された命令コードをオブジェクトファイルに出力する（ステップＳ３１３）。出力する順番及びバイトごとの配置は、ＢＵＦ［０］、ＢＵＦ［１］、ＢＵＦ［２］、ＢＵＦ［３］の順番で、リトルエンディアンに対応させてオブジェクトファイルの下位アドレス（即ち、ＬＳＢ）から出力する。

【００４０】ステップＳ３０５におけるコードサイズチェックの結果、変換された命令コードが１６ビット長であった場合は、一時保管バッファ５に前回変換分の命令コードが残存しているか否かをチェックする（ステップＳ３１０）。残存していなければ、１６ビット長命令コードの上位８ビットを一時保管バッファ５のＢＵＦ［３］に、下位８ビットをＢＵＦ［２］に保管する（ステップＳ３１１）。

【００４１】つぎに、ファイル入力部２においてソースファイルからつぎの一行のソースコードを読み出し（ステップＳ３０２）、同様にステップＳ３０５までの処理を行う。この命令が１６ビット長命令コードであった場合、一時保管バッファ５に前回変換分の命令コードが残存しているか否かをチェックするが（ステップＳ３１０）、この場合、ＢＵＦ［３］とＢＵＦ［２］に残存し

ているので、新たに変換した命令コードの上位８ビットをＢＵＦ［１］に、下位８ビットをＢＵＦ［０］に保存する（ステップＳ３１２）。そして、ファイル出力部７によって、一時保管バッファ５に格納された命令コードを、ＢＵＦ［０］、ＢＵＦ［１］、ＢＵＦ［２］、ＢＵＦ［３］の順番で、リトルエンディアンに対応させてオブジェクトファイルの下位アドレス（即ち、ＬＳＢ）から出力する（ステップＳ３１３）。

【００４２】ステップＳ３０５で、１６ビット長命令コードのつぎに読み出して変換された命令コードが３２ビット長命令であった場合には、一時保管バッファ５に前回変換分の命令コードが残存しているか否かをチェックするが（ステップＳ３０６）、この場合、ＢＵＦ［３］とＢＵＦ［２］に残存しているので、ステップＳ３０７へ進み、一時保管バッファ５のＢＵＦ［１］、ＢＵＦ［０］に１６ビットの無処理命令（ＮＯＰ）を追加保存する。そして、ファイル出力部７によって、一時保管バッファ５に格納された命令コードを、ＢＵＦ［０］、ＢＵＦ［１］、ＢＵＦ［２］、ＢＵＦ［３］の順番で、リトルエンディアンに対応させてオブジェクトファイルの下位アドレス（即ち、ＬＳＢ）から出力し（ステップＳ３０８）、続いて次の３２ビット長命令コードの一時保管バッファ５への保存（ステップＳ３０９）およびオブジェクトファイルへの出力を行う（ステップＳ３１３）。

【００４３】図５（ａ）は、一例としてソースコードが３２ビット長の命令コードＡ，１６ビット長の命令コードＢ，および１６ビット長の命令コードＣにこの順番でコード変換された場合の、リトルエンディアン用の各命令コードの一時保管バッファ５への格納状態、およびオブジェクトファイルへの出力状態を示す説明図である。図５（ｂ）は、一例としてソースコードが１６ビット長の命令コードＢ，３２ビット長の命令コードＡにこの順番でコード変換された場合の、各命令コードの一時保管バッファ５への格納状態、およびオブジェクトファイルへの出力状態を示す説明図である。

【００４４】プロセッサがリトルエンディアンで動作する場合にも、プロセッサがオブジェクトファイルからメモリ上にロードされた命令コードを０番地、４番地、６番地の順に実行していくが、図５（ａ）および（ｂ）からわかるように、オブジェクトファイルへの各命令コードの格納配置はリトルエンディアンの場合に命令コードの実行順に配置されるよう変換されるので、ソースファイルの命令の順番と命令の実行順が一致したものとなる。

【００４５】以上説明したようにアセンブラ処理を行うことでアセンブリ言語記述の同一ソースファイルを用いてもプロセッサにおいてどちらのエンディアンにおいても同一の処理を実行するオブジェクトファイルを生成することができ、ソフトウェアの生産性を改善することができる。

【００４６】実施の形態２．実施の形態１のバイエンディアン対応アセンブラは、３２ビット境界で１６ビット長命令に続く命令が１６ビット長命令であった場合は２つの命令を繋げて３２ビット分を変換した後にオブジェクトファイルに出力するものであったが、この実施の形態２のバイエンディアン対応アセンブラは、１６ビット命令を変換する際に残り半分の１６ビットを常にＮＯＰ命令とすることで処理の簡略化を行うものである。

【００４７】図６は、この発明の実施の形態２であるアセンブラの構成を示すブロック図である。図６に示すように、実施の形態２のバイエンディアン対応アセンブラは、ファイル入力部２と、構文解析部３と、コード変換部４と、コード変換テーブル１０と、一時保管バッファ５と、配置変換部６６と、ファイル出力部７とを備えている。ここで、ファイル入力部２と、構文解析部３と、コード変換部４と、コード変換テーブル１０と、一時保管バッファ５と、ファイル出力部７は、実施の形態１のアセンブラと同様であるため、説明を省略する。配置変換部６６は、一時保管バッファ５に１６ビット長命令コードが保存された場合に、ＮＯＰ命令を追加して動作するエンディアンに応じて１６ビット長命令コードとＮＯＰ命令の配置を変換するものである。

【００４８】つぎに、このように構成された実施の形態２のバイエンディアン対応アセンブラによるビッグエンディアン用の命令コード生成処理手順について説明する。図７は、ビッグエンディアン用命令コード生成の処理手順を示すフローチャートである。

【００４９】アセンブラ１は、実施の形態１のビッグエンディアン用命令コード生成処理と同様に、一時保管バッファ５の初期化（ステップＳ６０１）、ソースファイルから一行読出し（ステップＳ６０２）、構文解析（ステップＳ６０３）、コード変換を行う（ステップＳ６０４）。そして配置変換部６６では、更に以下のリトルエンディアン用の命令コード生成のための配置変換処理を行う。まず、コード変換された命令コードのサイズをチェックする（ステップＳ６０５）。

【００５０】命令コードが３２ビット長であった場合には、コード変換された命令コードの上位バイトから１バイトずつ４バイト分を、一時保管バッファ５にＢＵＦ［０］、ＢＵＦ［１］、ＢＵＦ［２］、ＢＵＦ［３］の順で格納する（ステップＳ６０６）。そして、ファイル出力部７によって、一時保管バッファ５に格納された命令コードを、ＢＵＦ［０］、ＢＵＦ［１］、ＢＵＦ［２］、ＢＵＦ［３］の順番で、ビッグエンディアンに対応させてオブジェクトファイルの下位アドレス（即ち、ＭＳＢ）から出力する（ステップＳ６１０）。

【００５１】ステップＳ６０５におけるコードサイズのチェックの結果、変換された命令コードが１６ビット長であった場合は、１６ビット長命令コードの上位８ビッ

トを一時保管バッファ５のＢＵＦ［０］に、下位８ビットをＢＵＦ［１］に保管する（ステップＳ６０７）。そして、一時保管バッファ５のＢＵＦ［２］、ＢＵＦ［３］に１６ビットの無処理命令（ＮＯＰ）を追加保存する（ステップＳ６０８）。更に、ＮＯＰ命令を保存するバッファのＢＵＦ［２］の最上位ビットに「１」を設定する（ステップＳ６０９）。ＢＵＦ［２］の最上位ビットは、図１３の動作を行うプロセッサが命令コードを実行するときに、読み込んだ２つの１６ビット長の命令を並列に実行するか否かを判断する際の並列実行ビットに対応するものである。かかる最上位ビットを「１」に設定することにより、ＢＵＦ［０］とＢＵＦ［１］に保存された１６ビット長命令とＢＵＦ［２］とＢＵＦ［３］に保存されたＮＯＰ命令とがプロセッサによって並列に実行されることになる。

【００５２】そして、ファイル出力部７によって、一時保管バッファ５に格納された命令コードを、ビッグエンディアンに対応するため、ＢＵＦ［０］、ＢＵＦ［１］、ＢＵＦ［２］、ＢＵＦ［３］の順番で、ビッグエンディアンに対応させてオブジェクトファイルの下位アドレス（即ち、ＭＳＢ）から出力する（ステップＳ６１０）。

【００５３】つぎに、バイエンディアン対応アセンブラによるリトルエンディアン用の命令コード生成処理手順について説明する。図８は、リトルエンディアン用命令コード生成の処理手順を示すフローチャートである。

【００５４】アセンブラ１は、上述のビッグエンディアン用命令コード生成処理と同様に、一時保管バッファ５の初期化（ステップＳ７０１）、ソースファイルから一行読出し（ステップＳ７０２）、構文解析（ステップＳ７０３）、コード変換を行う（ステップＳ７０４）。そして配置変換部６６では、更に以下のリトルエンディアン用の命令コード生成のための配置変換処理を行う。

【００５５】まず、コード変換された命令コードのサイズをチェックする（ステップＳ７０５）。命令コードが３２ビット長であった場合には、コード変換された命令コードの下位バイトから１バイトずつ４バイト分を、一時保管バッファ５にＢＵＦ［０］、ＢＵＦ［１］、ＢＵＦ［２］、ＢＵＦ［３］の順で格納する（ステップＳ７０６）。そして、ファイル出力部７によって、ＢＵＦ［０］、ＢＵＦ［１］、ＢＵＦ［２］、ＢＵＦ［３］の順番で、リトルエンディアンに対応させてオブジェクトファイルの下位アドレス（即ち、ＬＳＢ）から一時保管バッファ５に格納された命令コードをオブジェクトファイルに出力する（ステップＳ７１０）。

【００５６】ステップＳ７０５におけるコードサイズのチェックの結果、コード変換された命令コードが１６ビット長であった場合は、１６ビット長命令コードの上位８ビットを一時保管バッファ５のＢＵＦ［３］に、下位８ビットをＢＵＦ［２］に保管する。（ステップＳ７０７）。そして、一時保管バッファ５のＢＵＦ［１］、ＢＵＦ［０］に１６ビットの無処理命令（ＮＯＰ）を追加保存する（ステップＳ７０８）。更に、１６ビット長命令コードを保存するバッファのＢＵＦ［２］の最上位ビットに「１」を設定する（ステップＳ７０９）。このＢＵＦ［２］の最上位ビットは、ビッグエンディアン用の処理と同様に、並列実行ビットに対応するものである。かかる最上位ビットを「１」に設定することにより、ＢＵＦ［３］とＢＵＦ［２］に保存された１６ビット長命令とＢＵＦ［１］とＢＵＦ［０］に保存されたＮＯＰ命令とがプロセッサによって並列に実行されることになる。

【００５７】そして、ファイル出力部７によって、一時保管バッファ５に格納された命令コードを、ＢＵＦ［０］、ＢＵＦ［１］、ＢＵＦ［２］、ＢＵＦ［３］の順番で、リトルエンディアンに対応させてオブジェクトファイルの下位アドレス（即ち、ＬＳＢ）から出力する（ステップＳ７１０）。

【００５８】このように、実施の形態２のアセンブラでは、全ての１６ビット長命令を３２ビットの境界に配置するようにしているので、リトルエンディアン動作時において１６ビット長命令がアドレスに逆行した実行動作となることを無視することができ、また逆アセンブルでメモリイメージとニーモニックの対応表示を行った場合にはコーディングした手順がそのまま出力されるため検証には都合がよくソフトウェア開発の効率化を行うことができる。また、並列実行ビットに対応するビットを「１」に設定しているので、１６ビット長命令と１６ビット長のＮＯＰ命令とを並列実行させることができ、ＮＯＰ命令を追加したことによるプロセッサの実行時間の増加を少なくすることができる。

【００５９】実施の形態３．実施の形態１および実施の形態２のバイエンディアン対応アセンブラは、ビッグエンディアンとリトルエンディアンの動作モードにおいて同一のソースファイルを用いることができる利点があるが、リトルエンディアンで動作したときのプロセッサでは命令実行におけるアドレスの逆行現象は避けられない。そこで、実施の形態３のバイエンディアン対応アセンブラでは、１６ビット長命令の全ての組み合わせについて複合命令のニーモニックを定義し、この複合命令で記述されたソースファイルを命令コードに変換することによりアドレスの逆行を完全に防止するものである。

【００６０】図９は、実施の形態３のアセンブラの構成を示すブロック図である。実施の形態３のアセンブラ１は、図９に示すように、ファイル入力部２と、構文解析部３と、コード変換部９０４と、コード変換テーブル９１０と、ファイル出力部７とを備えている。ここで、ファイル入力部２と、構文解析部３と、ファイル出力部７は、従来のアセンブラと同様の構成であるため説明を省略する。

【００６１】図１０は、実施の形態３のアセンブラで使用される複合命令の定義例を示すブロック図である。「複合命令」とは、２つの１６ビット長の命令を組み合わせた命令である。図１０において、左列および中央列のブロックは既存の１６ビット長命令のニーモニック表記８であり、右列のブロックは既存の１６ビット長命令を２つ組み合わせて２命令分の処理内容を表すニーモニック表記９である。図１０に示すように、複合命令のニーモニック表記９は、２つの１６ビット長命令のすべての組み合わせについて定義されている。

【００６２】実施の形態３のアセンブラは、このような複合命令で記述されたソースファイルから命令コード（機械語）を生成するものであり、このためコード変換部９０４は、ソースファイルから読み込んだ複合命令を、コード変換テーブル９１０を参照して、対応する複合命令の命令コード（機械語）に変換するようになっている。コード変換テーブル９１０は、図１０の定義例に示した複合命令のニーモニック表記９と複合命令の命令コードとを、前述した図１４に示す形式でように対応づけたものである。

【００６３】図１１は、実施の形態３のアセンブラによるアセンブラ処理手順を示すフローチャートである。図１１に示すように実施の形態３のアセンブラでは、まずバッファを初期化する（ステップＳ１１０１）。そして、ファイル入力部２によって、複合命令で記述されたソースファイルから複合命令を一行読み込み（ステップＳ１１０２）、読み込んだ複合命令を構文解析部３によって解析する（ステップＳ１１０３）。つぎに、コード変換部９０４によって、コード変換テーブル９１０を参照して複合命令のニーモニックを３２ビット長の命令コード（機械語）に変換し（ステップＳ１１０４）、変換された機械語をバッファへ保存する（ステップＳ１１０５）。バッファに保存された機械語は、ファイル出力部７によって、エンディアンに従ってオブジェクトファイルにを生成する（ステップＳ１１０６）。このようなステップＳ１１０２からステップＳ１１０６までの処理を、ソースファイルに記述されているすべての行の複合命令について繰り返し行う。

【００６４】このように実施の形態３のアセンブラでは、図１０に示すような複合命令を定義して、この複合命令のニーモニック表記と命令コードとを対応づけるコード変換テーブル９１０を使用してコード変換部９０４により命令コードを生成しているので、生成される命令コードは全て３２ビット長となる。このため、プロセッサのリトルエンディアン動作における命令の逆転実行を確実になくすことができ、また生成される命令コードのコード量の最適化を図ることが可能となる。

【００６５】

【発明の効果】以上説明したように、この発明によれば、プロセッサがビッグエンディアンとリトルエンディアンのどちらのエンディアンで動作する場合でも、アセンブリ言語記述の同一ソースファイルを用いて同一の処理を実行するオブジェクトファイルを生成することができ、ソフトウェアの生産性を改善することができるという効果を奏する。

【００６６】つぎの発明によれば、全ての１６ビット長命令を３２ビットの境界に配置して、リトルエンディアン動作時において１６ビット長命令がアドレスに逆行した実行動作となることを無視することができるという効果を奏する。また、逆アセンブルでメモリイメージとニーモニックの対応表示を行った場合にはコーディングした手順がそのまま出力されるため、検証には都合がよくソフトウェア開発の効率化を図れるという効果を奏する。

【００６７】つぎの発明によれば、プロセッサのリトルエンディアン動作時における命令の逆転実行を確実になくすことができ、また、コード量の最適化を図れるという効果を奏する。

【００６８】つぎの発明によれば、プロセッサがビッグエンディアンとリトルエンディアンのどちらのエンディアンで動作する場合でも、アセンブリ言語記述の同一ソースファイルを用いて同一の処理を実行するオブジェクトファイルを生成することができ、ソフトウェアの生産性を改善することができるという効果を奏する。

【００６９】つぎの発明によれば、全ての１６ビット長命令を３２ビットの境界に配置して、リトルエンディアン動作時において１６ビット長命令がアドレスに逆行した実行動作となることを無視することができるという効果を奏する。また、逆アセンブルでメモリイメージとニーモニックの対応表示を行った場合にはコーディングした手順がそのまま出力されるため、検証には都合がよくソフトウェア開発の効率化を図れるという効果を奏する。

【００７０】つぎの発明によれば、プロセッサのリトルエンディアン動作時における命令の逆転実行を確実になくすことができ、また、コード量の最適化を図れるという効果を奏する。

【００７１】つぎの発明によれば、上記の発明のいずれか一つの動作をコンピュータによって実行することができるという効果を奏する。

【図面の簡単な説明】

【図１】　この発明の実施の形態１および実施の形態２のアセンブラの構成を示すブロック図である。

【図２】　図１に示した実施の形態１のアセンブラによるビッグエンディアン用命令コード生成処理手順を示すフローチャートである。

【図３】　図１に示した実施の形態１のアセンブラによるリトルエンディアン用命令コード生成処理手順を示すフローチャートである。

【図４】　図４（ａ）は、図１に示した実施の形態１の

アセンブラにおいて、ビッグエンディアン用命令コード生成時の３２ビット長の命令コードA，１６ビット長の命令コードB，および１６ビット長の命令コードCの一時保管バッファへの格納状態、およびオブジェクトファイルへの出力状態を示す説明図である。図４（ｂ）は、図１に示した実施の形態１のアセンブラにおいて、ビッグエンディアン用命令コード生成時の１６ビット長の命令コードB，３２ビット長の命令コードAの一時保管バッファへの格納状態、およびオブジェクトファイルへの出力状態を示す説明図である。

【図５】　図５（ａ）は、図１に示した実施の形態１のアセンブラにおいて、リトルエンディアン用命令コード生成時の３２ビット長の命令コードA，１６ビット長の命令コードB，および１６ビット長の命令コードCの一時保管バッファへの格納状態、およびオブジェクトファイルへの出力状態を示す説明図である。図５（ｂ）は、図１に示した実施の形態１のアセンブラにおいて、リトルエンディアン用命令コード生成時の１６ビット長の命令コードB，３２ビット長の命令コードAの一時保管バッファへの格納状態、およびオブジェクトファイルへの出力状態を示す説明図である。

【図６】　この発明の実施の形態２のアセンブラの構成を示すブロック図である。

【図７】　図６に示したアセンブラによるビッグエンディアン用命令コード生成処理手順を示すフローチャートである。

【図８】　図６に示した実施の形態２のアセンブラによるリトルエンディアン用命令コード生成処理手順を示すフローチャートである。

【図９】　この発明の実施の形態３のアセンブラの構成を示すブロック図である。

【図１０】　図９に示した実施の形態３のアセンブラにおいて使用される複合命令の定義例を示すブロック図である。

＊【図１１】　図９に示した実施の形態３のアセンブラによるアセンブラ処理手順のフローチャートである。

【図１２】　図１、図６および図９に示したアセンブラを適用するプロセッサの命令の種類を示す図である。

【図１３】　図１、図６および図９に示したアセンブラを適用するプロセッサの命令実行手順を示すフローチャートである。

【図１４】　図１および図６に示したアセンブラのコード変換部で参照するコード変換テーブルの説明図である。

【図１５】　機械語に変換した１６進数コードのメモリ配置を示す図である。

【図１６】　ビッグエンディアンにおける機械語のメモリ配置を示す説明図である。

【図１７】　リトルエンディアンにおける機械語のメモリ配置を示す説明図である。

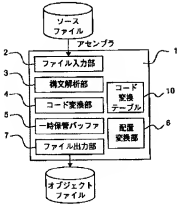
【図１８】　従来のアセンブラの構成を示すブロック図である。

【図１９】　従来のアセンブラにおけるアセンブラ処理手順のフローチャートである。

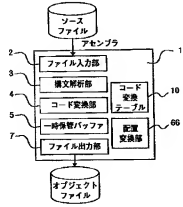
【符号の説明】

１　アセンブラ、２　ファイル入力部、３　構文解析部、４　コード変換部、５　一時保管バッファ、６　配置変換部、７　ファイル出力部、８　ニーモニック表記、９　複合命令のニーモニック表記、１０　コード変換テーブル、１１アセンブラ、１２　ファイル入力部、１３　構文解析部、１４　コード変換部、１７　ファイル出力部、１８　ニーモニック、１９　機械語、２０　ニーモニック、２１　機械語（命令コード）、２２　アドレス、２３　命令コードの配置（ビッグエンディアン）、２４　命令コードの配置（リトルエンディアン）、６６配置変換部、９０４　コード変換部、９１０　コード変換テーブル。

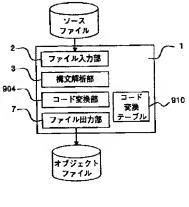
【図１】



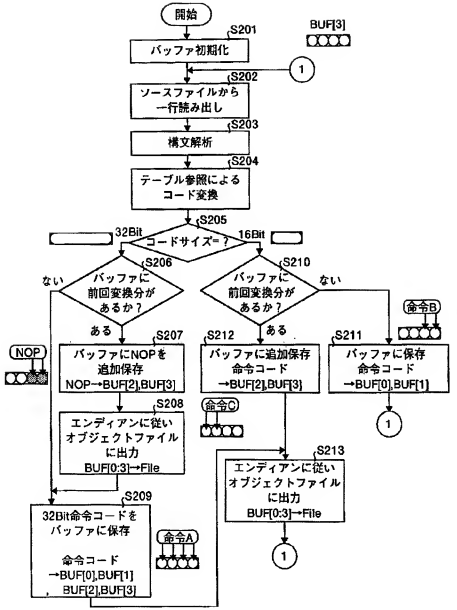
【図６】



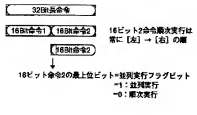
【図９】



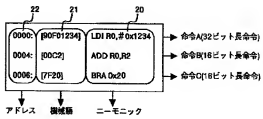
【図２】


開始
S201
BUF[3]
バッファ初期化
1
S202
ソースファイルから
一行読み出し
S203
構文解析
S204
テーブル参照による
コード変換
S205
32Bit
コードサイズ=？
16Bit
S206
バッファに
前回変換分が
あるか？
ない
ある
S210
バッファに
前回変換分が
あるか？
ない
ある
命令B
NOP
S207
バッファにNOPを
追加保存
NOP→BUF[2],BUF[3]
S212
バッファに追加保存
命令コード
→BUF[2],BUF[3]
S211
バッファに保存
命令コード
→BUF[0],BUF[1]
1
S208
エンディアンに従い
オブジェクトファイル
に出力
BUF[0:3]→File
命令C
S213
エンディアンに従い
オブジェクトファイル
に出力
BUF[0:3]→File
1
S209
32Bit命令コードを
バッファに保存
命令コード
→BUF[0],BUF[1]
BUF[2],BUF[3]
命令A

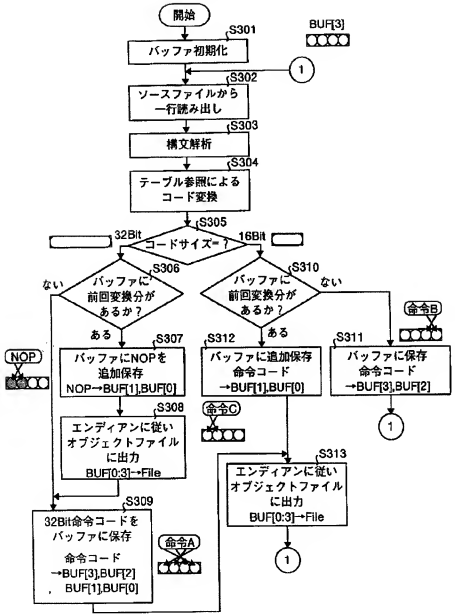
【図１２】


32Bit長命令
16Bit命令1
16Bit命令2
16Bit命令2
16ビット2命令順次実行は
常に［左］→［右］の順
16Bit命令2の最上位ビット=並列実行フラグビット
=1：並列実行
=0：順次実行

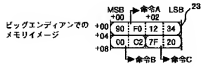
【図１５】


22
21
20
0000:
[90F01234]
LDI R0,#0x1234
命令A(32ビット長命令)
0004:
[00C2]
ADD R0,R2
命令B(16ビット長命令)
0006:
[7F20]
BRA 0x20
命令C(16ビット長命令)
アドレス
機械語
ニーモニック

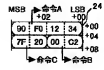
【図３】


開始
S301
バッファ初期化
BUF[3]
1
S302
ソースファイルから
一行読み出し
S303
構文解析
S304
テーブル参照による
コード変換
S305
32Bit
コードサイズ＝？
16Bit
S306
バッファに
前回変換分が
あるか？
ない
ある
S310
バッファに
前回変換分が
あるか？
ない
ある
命令B
NOP
S307
バッファにNOPを
追加保存
NOP→BUF[1],BUF[0]
S312
バッファに追加保存
命令コード
→BUF[1],BUF[0]
S311
バッファに保存
命令コード
→BUF[3],BUF[2]
1
命令C
S308
エンディアンに従い
オブジェクトファイル
に出力
BUF[0:3]→File
S313
エンディアンに従い
オブジェクトファイル
に出力
BUF[0:3]→File
1
S309
32Bit命令コードを
バッファに保存
命令コード
→BUF[3],BUF[2]
BUF[1],BUF[0]
命令A

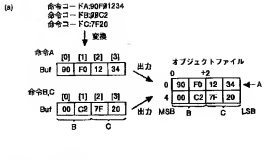
【図１６】


ビッグエンディアンでの
メモリイメージ
MSB
+00
命令A
+02
LSB
23
+00
90
F0
12
34
+04
00
C2
7F
20
+08
命令B
命令C

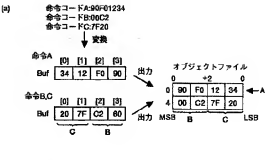
【図１７】


リトルエンディアンでの
メモリイメージ
MSB
+00
命令A
+02
LSB
24
+00
90
F0
12
34
+04
7F
20
00
C2
+08
命令C
命令B

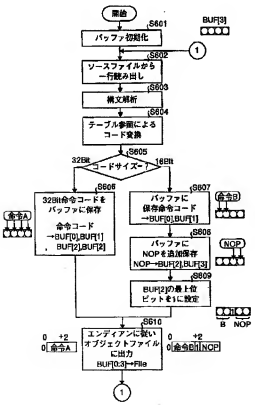
【図４】


(a)
命令コードA:90F81234
命令コードB:8BC2
命令コードC:7F20
変換
命令A
Buf
出力
オブジェクトファイル
命令B,C
MSB
LSB
(b)
命令コードB:00C2
命令コードA:90F01234
NOP

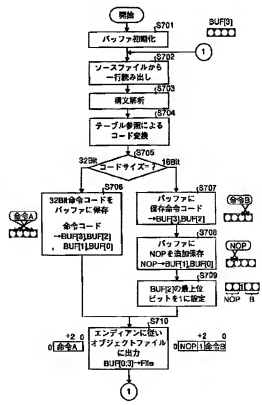
【図５】


(a)
命令コードA:90F01234
命令コードB:00C2
命令コードC:7F20
変換
命令A
命令B,C
出力
オブジェクトファイル
MSB
LSB
(b)
命令コードB:00C2
命令コードA:90F01234
NOP

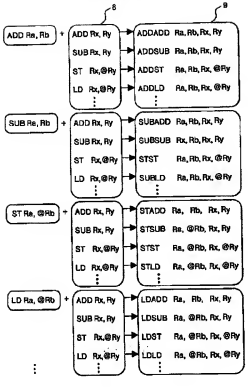
【図７】


開始
S601
BUF[3]
バッファ初期化
S602
ソースファイルから
一行読み出し
S603
構文解析
S604
テーブル参照による
コード変換
S605
32Bit
コードサイズ
16Bit
S606
32Bit命令コードを
バッファに保存
命令コード
→BUF[0],BUF[1]
BUF[2],BUF[3]
S607
バッファに
保存命令コード
→BUF[0],BUF[1]
S608
バッファに
NOPを追加保存
NOP→BUF[2],BUF[3]
S609
BUF[3]の最上位
ビットをHに設定
S610
エンディアンに従い
オブジェクトファイル
に出力
BUF[0:3]→File
命令A
命令B
NOP
H
L
0
+2
+2 0
1

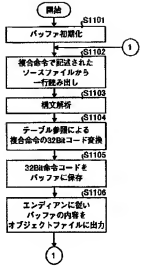
【図８】


開始
S701
BUF[3]
バッファ初期化
S702
ソースファイルから
一行読み出し
S703
構文解析
S704
テーブル参照による
コード変換
S705
32Bit
コードサイズ
16Bit
S706
32Bit命令コードを
バッファに保存
命令コード
→BUF[3],BUF[2]
BUF[1],BUF[0]
S707
バッファに
保存命令コード
→BUF[3],BUF[2]
S708
バッファに
NOPを追加保存
NOP→BUF[1],BUF[0]
S709
BUF[2]の最上位
ビットをHに設定
S710
エンディアンに従い
オブジェクトファイル
に出力
BUF[0:3]→File
命令A
命令B
NOP
H
L
0
+2
+2 0
1

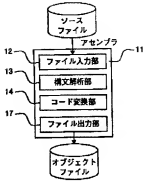
【図１０】



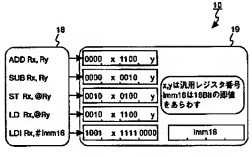
【図１１】



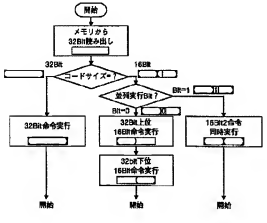
【図１８】



【図１４】



【図１３】



【図１９】